Fermi National Accelerator Laboratory

TM-1707

Table of Tables - A Database Design Tool for SYBASE

Bruce C. Brown, Karen Coulter, Henry D. Glass, Richard Glosson, Raymond W. Hanft, David J. Harding, Kelley Trombly-Freytag, Dana G. C. Walbridge and David B. Wallis
Fermi National Accelerator Laboratory
P.O. Box 500
Batavia, Illinois 60510

Michael E. Allen
SSC Laboratory
Dallas, Texas 75237

January 4, 1991

Operated by Universities Research Association Inc. under contract with the United States Department of Energy

TABLE OF TABLES – A DATABASE DESIGN TOOL FOR SYBASE

Bruce C. Brown, Karen Coulter, Henry D. Glass,
Richard Glosson, Raymond W. Hanft, David J. Harding,
Kelley Trombly-Freytag, Dana G. C. Walbridge, David B. Wallis
Fermi National Accelerator Laboratory *
P.O. Box 500
Batavia, Illinois 60510

Michael E. Allen
*SSC Laboratory**
2550 Beckleymeade Avenue
Dallas, TX 75237

January 4, 1991

Abstract

The 'Table of Tables' application system captures in a set of SYBASE tables the basic design specification for a database schema. Specification of tables, columns (including the related defaults and rules for the stored values) and keys is provided. The feature which makes this application specifically useful for SYBASE is the ability to automatically generate SYBASE triggers. A description field is provided for each database object. Based on the data stored, SQL scripts for creating complete schema including the tables, their defaults and rules, their indexes, and their SYBASE triggers, are written by TOT. Insert, update and delete triggers are generated from TOT to guarantee integrity of data relations when tables are connected by single column foreign keys. The application is written in SYBASE's APT-SQL and includes a forms based data entry system. Using the features of TOT we can

*Operated by the Universities Research Association under contract with the U. S. Department of Energy

create a complete database schema for which the data integrity specified by our design is guaranteed by the SYBASE triggers generated by TOT.

1 Introduction

To provide a systematic tool for defining and creating SYBASE database schemas, we have created an application – Table of Tables (TOT)[1]. With it we create and document database schema for a variety of SYBASE[2] applications. TOT supports both the usual features of a relational model database and many SYBASE-specific extensions. Existing computer assisted software engineering tools provide no SYBASE-specific features and in addition are costly and contain features not directly applicable to the SYBASE applications we choose to develop.

A relational database schema consists of a set of (usually related) multi-column tables. Values to be stored in columns can be limited by use of Rules and Defaults (see Appendix G). A key is a column or set of columns whose contents are guaranteed to be unique among all rows in a table. SYBASE recognizes three types of keys: primary keys, common keys and foreign keys. To this we add (from the literature on Relational Databases) the concept of an alternate key. A primary key is the column or set of columns which serves as the principle unique specifier for the table. We choose to implement this for most of our tables using a special column consisting of a serial number. This is sufficient for making the row unique for retrieval, but since the serial number contains no data, we want something other than the serial number to uniquely define the row. That combination which uniquely defines the data in the row we call an alternate key. To facilitate joining information from different tables, SYBASE maintains a table which lists the keys shared between two tables – the common keys. A foreign key is a column or set of columns in a table which can only assume values of an existing primary key in another table. In Appendix F the support for keys within TOT is discussed more fully.

TOT consists of three parts:

[1]Table of Tables (TOT) user interface application was designed and implemented by Richard Glosson. This application includes the data entry application and the automatic generation of SQL code, which uses SYBASE to create tables, keys, indexes, triggers, datatypes, rules and defaults.

[2]SYBASE, APT-SQL, APT-Execute, APT Workbench, Transact-SQL and Data Workbench are registered trademarks of Sybase, Inc.

1. A schema for describing a SYBASE relational database schema. (The information is stored in a set of related SYBASE tables.)

2. A data entry application to assist in filling the tables defined for TOT

3. Application code which allows

   - TOT to create SQL scripts which can be executed to create the database system which has been described in TOT tables
   - TOT to create datatypes which have been defined in the TOT schema

Complete prescriptions for a large class of SYBASE database schema can be captured in TOT and the required database can be created using SQL scripts output by TOT. When required, the scripts written by TOT can be modified with additional trigger[3] or other SQL code. The complete data integrity features provided by SYBASE through its trigger mechanisms was a major factor in our choice of the SYBASE database management system. By developing TOT as an application specific to our needs, we can take full advantage of these features. They make it possible to preserve data integrity while using only a general tool (such as the SYBASE Data Workbench) for a large variety of database entry requirements, thus avoiding creation of some special data entry applications.

The feature missing from Table of Tables is a graphic display of table design. We have chosen to provide graphic descriptions manually, utilizing a graphic representation suggested in the text *A Visual Introduction to SQL*[1]. We adapted it to capture additional features we find useful in our specific design. While our graphic representation could be drawn using any convenient graphical drawing package, we have used the IDRAW package which is part of the Interviews[2][3] system, a public domain product available for UNIX machines. The capture of these visual descriptions is thus a manual operation uncoupled from the TOT application itself.

[3]SYBASE provides the stored procedure mechanism to store SQL procedures within the database itself for execution by the database server. A Trigger is a special stored procedure designed to provide integrity by tying its execution to an insertion, update or deletion of a data row.

2 TOT Database Schema

In Fig 1 we illustrate the TOT schema [4] devised to store a description of a general database schema within SYBASE. A complete description of the TOT tables is included in Appendix B. It consists of a SYBASE report of our captured description of TOT stored in TOT. The dataowners, tables, columns, column_defaults, column_rules and foreignkeys tables capture the description of a database schema.

The datatypes, datatype_rules, and datatype_defaults tables allow the specification of application specific datatypes in terms of the SYBASE datatypes. For example, we define a serial number type (sntype) to be integer. This allows us to indicate its use by its name now, and in the future we could implement a different serial number scheme and need to make very limited changes in the contents of the TOT descriptions of our database schema.

Use of the overall hierarchical scheme available for grouping SYBASE database information (database, owner, table) is limited by the ability to share datatype definitions and to make joins and triggers across databases. In addition, some SYBASE administration is easier with a single database. A self-consistent back-up is only guaranteed at the database level. We have therefore chosen to store all the data for a project in a single SYBASE database. We desire to provide some structure to a database other than the user names that are typically associated with tables. We do that by establishing "dataowners", special "users" in the system which own a group of related tables. A single person has responsibility for each dataowner account. For example, TOT could label the dataowner of the TOT tables. Separate portions of the project with different dataowners will have their SYBASE entities stored and accessed using that dataowner information.

The columns of the dataowners table are the dataowner_name, description (why is this data stored separately), responsible_person (an individual who assumes responsibility for this portion of the schema being implemented) and entry_date.

The tables table lists the dataowner, table_name, and description along with a responsible_person, and entry_date. The table_name is to be unique across the entire database, independent of
dataowner_name. The columns table provides a complete description of the columns in all tables. The column_name, table_name, and the data type

[4] As discussed previously, it is drawn using the IDRAW package which captures it as a postscript file ready for printing.

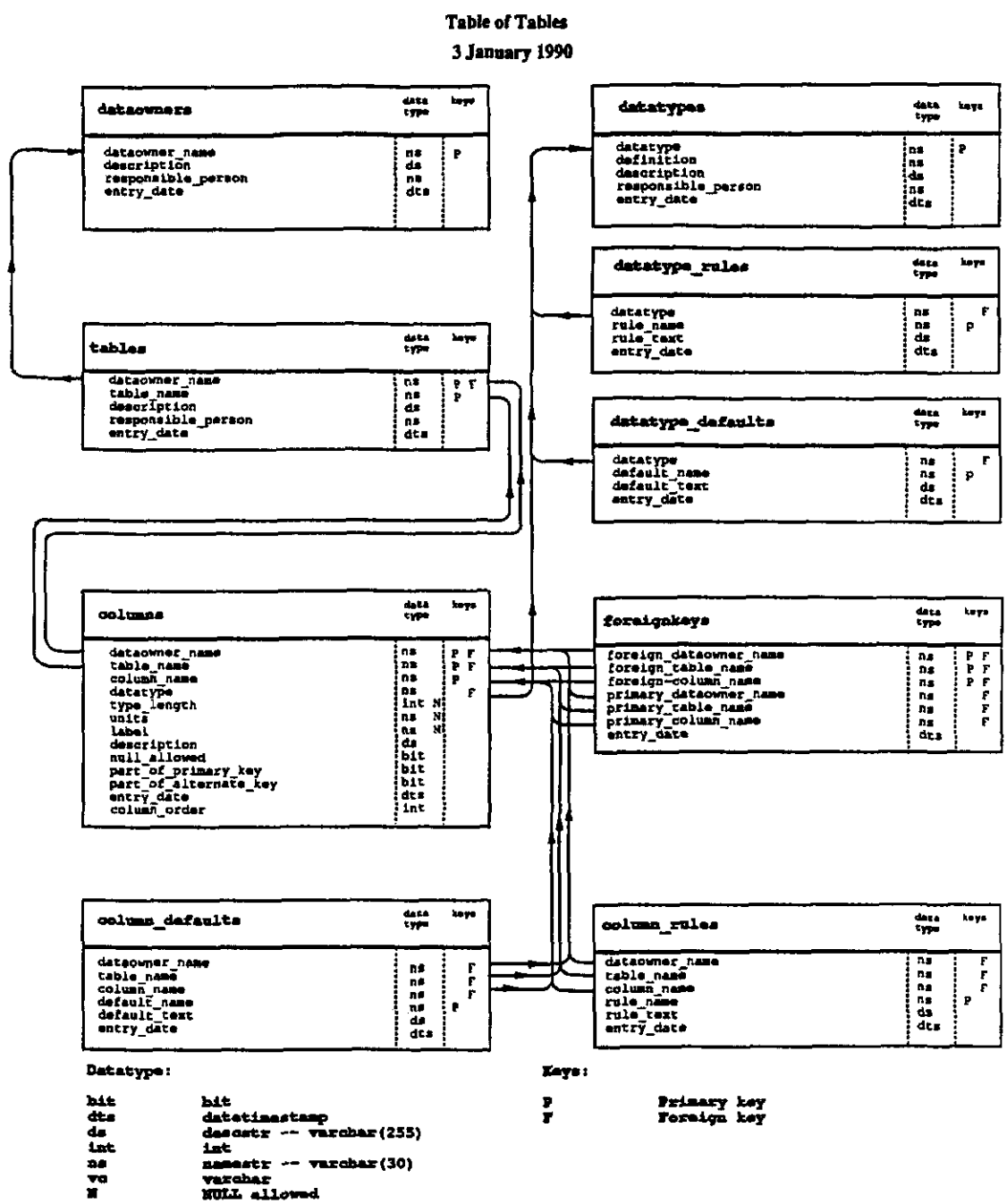

Figure 1: Database Schema for TOT

(and length if needed) are required.[5] We store whether the column is part of the primary key, part of the alternate key, and whether NULL is allowed.

The column order provides a default order for *ad hoc* queries (as well as the column order for multicolumn keys). TOT requires the designer to specify the column order. It is left to the database designer to make these sequential and meaningful. The TOT application does not assign them or require them to be sequential. An ordered list of columns will be provided based on the values in the column order field. Duplicate values will assume an order which is not guaranteed by the design.

In addition to the minimum information required to define the schema, we include information on the unit of measure for the column and a label which can be used in place of the column name for labeling a report. It is our intention to utilize SI Units (System International), but that restriction must be implemented through the contents one builds into the units column rather than through the design of TOT itself.

The column_defaults table allows one to specify a default value for a column as prescribed in the SYBASE documentation. Similarly, the column_rules table provides for capture of SYBASE rules. Since Rules and Defaults can be bound to user defined datatypes as well as to columns, TOT provides for the capture of datatype_rules and datatype_defaults.[6] For a complete description of the columns table, see Appendix B. See Appendix G for a discussion of defaults and rules.

To enforce referential integrity for a database schema in SYBASE one can create triggers based on foreign keys. TOT's foreignkeys table provides a place to specify foreign keys when the key is a single column. The current TOT requires that the dataowner, table_name, and column_name must be specified for both the primary and foreign key (primary_dataowner, primary_table_name, primary_column_name, foreign_dataowner, foreign_table_name, and foreign_column_name)[7]. From this information TOT writes insert and update triggers which guarantee that a row may not be

[5] Dataowner is also included, a relic of the days before we required table names to be unique within the database, instead of only requiring uniqueness within the dataowner.

[6] The names of the rules and defaults must be unique in the database so TOT enforces uniqueness for column_rules and datatype_rules and for column_defaults and datatype_defaults.

[7] The redundancies are a combination of historical artifact and accommodation of coding ease. The dataowners are not needed if the table_names are required to be unique in the database. The primary column name is not needed since we follow the usual relational requirement that a foreign key point to a primary key, and there is only one primary key for any table.

added or changed unless the value of each foreign key matches a value in the primary key of the corresponding table. The delete triggers on the primary keys are written which guarantee that they will not be deleted while rows of other tables point to them as a foreign key. Note that more than one foreign key in a table is allowed to point to the primary key of another table[8].

By utilizing a serial number[9] as the primary key in most tables, and by utilizing that for most of our foreign keys, the foreignkey table of TOT satisfies more than 95% of our requirements for triggers in an absolutely straightforward fashion.

The SQL Permissions model which is incorporated within TOT is a simple one based on a set of groups. The initial implementation of TOT simply provides that various permissions (INSERT, UPDATE, DELETE) be granted for various predefined SYBASE groups. The permissions granted are the same for all tables and the APT-SQL code which defines the SQL script-writing section of TOT has this structure coded into it (hard-coded).

The indexing scheme for a database may have substantial complications to provide suitably responsive behavior for both insert and query. TOT provides only a "natural" start on the complete index scheme. Indices for tables created by TOT are, by default, created for primary, alternate and foreign keys. Additional or alternative indices can be created by the Database Administrator. By default, the SQL code created by TOT will provide

1. A clustered, unique index on the primary key
2. A unique index on the alternate key
3. An index (non-unique, non-clustered) on the foreign key

[8] An alternate foreign key possibility which could also be easily coded allows a foreign key with an arbitrary number of columns but with only one foreign key from a given table allowed to point at any one primary table. Our systems require multiple foreign keys to the same primary and our reliance on serial numbers minimizes the need for multi-column foreign keys. Support for both multi-column foreign keys and multiple occurrences in a single table is straight-forward but sufficiently tedious that it was decided to defer implementation.

[9] The creation of serial numbers for MTF SYBASE databases is handled by a serial number generator implemented as a SYBASE stored procedure, allowing the application programs to avoid consideration of this problem. These serial numbers are unique within the database, not just within a specific table.

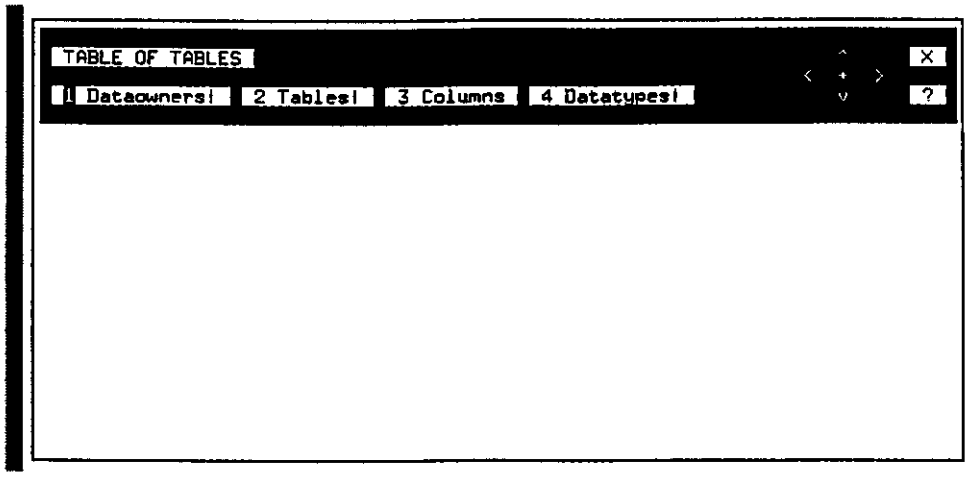

Figure 2: The Top Level Menu for the TOT Application

3 TOT Data Entry Application

Utilizing APT Workbench from SYBASE[10], we have built applications based on the tables described in the previous section to capture database designs. In this section we will briefly describe the functionality of that system. A step by step description of its use is available in Appendix D.

Figure 2 illustrates the screen view of the TOT main menu. From here we can choose to view or enter information into the dataowners, tables, columns or datatypes tables using a standard SYBASE type of display. Features for FINDing and SCROLLING assist in manipulating the displays to view entered or stored data. From each of these displays, we can either return to the main menu or move down the natural hierarchy to the next table, carrying along the environment of what we have just specified. One reaches the columns_defaults and columns_rules and foreignkeys table displays from the columns display. Similarly, we reach the datatype_defaults and datatype_rules via the datatype display.

[10] APT Workbench is a 4GL with a screen manipulation language that interfaces with the database.

4 Writing Table Definition SQL from TOT

The SQL-writing portion of TOT can be invoked for any tables that have been entered into the columns form, regardless of whether they have been saved yet. When the information needed for the specification of a system of tables is defined by entering data for dataowners, tables, columns and their keys, one can invoke the SQL-writing (BUILD) portion of TOT to provide SQL script files in your current working directory. These scripts can be executed to create tables and triggers. The created files contain SQL commands in ASCII so they can be viewed (or modified) with a text editor.[11]

BUILD commands are provided by TOT to create the output SQL scripts. The SQL to create a single table (including keys and indices) is provided by one BUILD command. The triggers scripts which apply to this table are created with an additional two BUILD commands. Datatypes are created by SYBASE directly from a CREATE command. Some applications may require additional SQL code for complete definition of the relationships specified by the designer. A detailed description of the code produced by TOT for creating table definitions and triggers from the information stored by TOT is contained in Appendix E. Examination of that information will allow one to understand the details of the relationship between the table entries and the SQL code generated.

A few guidelines are useful to describe how one employs TOT output and maintains a complete set of database descriptions. We emphasize that SYBASE allows flexibility for defining triggers which maintain data integrity with a complexity far greater than is provided in TOT. Thus in addition to the files provided which contain SQL for creating tables, their keys, and triggers relating tables, one might write more complex trigger code which could then be incorporated with the TOT output in order to create a complete trigger system.

Although the BUILD portions of TOT are available to all users and reading the code generated may clarify some design issues for users, we believe that a single Database Administrator should usually execute all of the scripts output by TOT. The trigger scripts, in particular, affect the entire database and must be carefully integrated.

[11] The variant of SQL which is created is SYBASE Transact-SQL. This is the SYBASE implementation of the SQL standard with enhancements, to be distinguished from the APT-SQL version. One can execute Transact-SQL scripts using the SYBASE isql application.

5 Using TOT

5.1 New Applications and Data Entry

The appendices contain specific instructions for TOT. We will describe here the general pattern for use.

Suppose we wish to add a new SYBASE-based application involving several new tables and several existing tables which have been previously captured in TOT. We begin by deciding to create a new dataowner category or use an existing one. We add any new datatypes specific to our application. Then we enter the table names and descriptions into the tables table. For each table we complete the columns table and the foreign keys table if required. We enter any needed rules and defaults.

With the data captured, we may then proceed to build the SQL code which will create the new tables with their associated keys. We will also build insert and update trigger code for each table, based in part on the foreign key information. When we seek to create the new delete triggers, an additional complication must be noted. New delete triggers should now be created for all tables related to the new tables through a foreign key[12]. In particular, if a column in one of the new tables has a foreign key pointing to a table described previously, we would want to be sure to update the triggers associated with deletion from that previous table. As part of the 'Fkey And Insert Trigger Code' option which creates the foreign keys for the new table, TOT will automatically create the delete trigger code for all of the associated primary tables to which the foreign keys point[13].

Let us specify the steps required to build a table from its TOT definition.

1. Use TOT procedure to create scripts to build table definition.
2. Use TOT procedure to create foreign keys and create insert and update trigger code.
3. Use TOT procedure to create delete trigger code.
4. For any features of this database schema which are not specified in TOT but can be supported in SYBASE, edit the required additional code into the trigger (or BUILDTABLE) scripts.

[12]This version of TOT does not include update trigger modification for the foreign key. Instead, by default, the update trigger forbids the modification of the primary key.

[13]If we have added extra code to the delete triggers of these tables, we need to add those changes to the SQL script files created by TOT *before* executing them, so as not to lose those changes.

5. If this is not a new application, save the data stored in existing versions of the table.

6. Create the new or revised tables by executing the SQL BUILDTABLE files.

7. Exercising due care for the existing tables, execute the scripts which create the trigger code.

8. If necessary, restore previously saved data into the newly created schema.

If we observe these simple rules, it is straightforward to ask SYBASE to produce and execute all required SQL code and have a complete new set of tables.

5.2 Changing Table Definitions

A few additional steps may be required when we have designed and brought into use a set of tables and then discover a requirement to revise those tables. We can capture the new design in TOT, but we must be aware that it will affect our ability to recreate the current situation. It is likely to be desirable to retain the SQL code that allowed us to create the present set of tables and triggers, so we first make an archival copy of that code. A standard code management system is an appropriate vehicle. We then modify the TOT description, create new SQL for defining the database and proceed as above, again preserving a copy of the critical SQL files. Depending upon the complexity of the TOT descriptions, we may wish to preserve a record of them, also, to avoid data entry errors if we want to reverse a TOT description change.

To preserve existing data stored in the table, we can extract the contents of the table using, for example, a bulk copy. Then, we use the SQL script for deleting and recreating the table. Finally we can bulk copy in the previous contents. For simple tables when no columns have been added, such a procedure is efficient and straightforward. Substantially more complicated efforts may be required in cases where large numbers of foreign keys carry us back perhaps several steps through a database. In this case, the procedure described might be sufficient but there are potential modifications for which additional trigger generation code will be required in place of the simple bulk copy before we can reinsert the data in the database.

6 Summary

Users of the SYBASE database may capture nearly complete descriptions of the database schema using the TOT application. They may wish to supplement the information they store in TOT with perhaps some additional trigger information for those triggers which go beyond the model supported in TOT and with modification of the indexes to optimize performance.

Complete prescription as well as documentation of the design is available from the reports stored in SYBASE. SQL code output in the TOT application can be used directly to build tables, triggers and keys for the desired schema.

A Appendix: Versions Status of TOT

This document describes TOT version 1.0 released in January 1991. It was developed initially under SYBASE Release 4.0 and updated to run under SYBASE Release 4.0.1, both running under SunOS Ver 4.0.3 on a Sun SPARC platform.

B Appendix: Complete reports of the TOT description of it database schema.

The following pages contain reports created from data stored in the columns table to describe the various TOT tables. This is followed by a table which describes the foreign keys required by the TOT schema. Most of these cannot be described by the TOT foreign keys table since they involve multicolumn keys.

Dbo.columns Report

dataowner name	table name	column order	column_name	datatype	type leng	null ?	pri key	alt key	description
tot	column_defaults	1	default_name	namestr	NULL	0	1	0	The unique name of this default (must be unique compared to other column defaults and the datatype defaults)
		2	dataowner_name	namestr	NULL	0	0	0	The section of the database that contains the columns the default applies to
		3	table_name	namestr	NULL	0	0	0	Name of the table containing a column this default is bound to
		4	column_name	namestr	NULL	0	0	0	Name of a column this default is bound to
		5	default_text	descstr	NULL	0	0	0	The Transact SQL code needed to create this default
		6	entry_date	datetimestamp	NULL	0	0	0	Date this default was entered or changed in TOT
	column_rules	1	rule_name	namestr	NULL	0	1	0	The unique name of this rule (must be unique compared to other column rules and the datatype rules)
		2	dataowner_name	namestr	NULL	0	0	0	The section of the database that contains the columns this rule applies to
		3	table_name	namestr	NULL	0	0	0	Name of the table containing a column this rule is bound to
		4	column_name	namestr	NULL	0	0	0	Name of a column this rule is bound to
		5	rule_text	descstr	NULL	0	0	0	The Transact SQL code needed to create this rule
		6	entry_date	datetimestamp	NULL	0	0	0	Date this rule was entered or changed in TOT
	columns	1	dataowner_name	namestr	NULL	0	1	0	The section of the database the table that this column belongs to resides under
		2	table_name	namestr	NULL	0	1	0	Name of the table this column belongs to
		3	column_name	namestr	NULL	0	1	0	Name of the column being described in TOT
		4	datatype	namestr	NULL	0	0	0	Datatype of the column, either built in or user defined

dataowner name	table name	column order	column_name	datatype	type leng	null ?	pri key	alt key	description
tot	columns	5	type_length	int	NULL	1	0	0	Length of datatype if built in. User defined types should not enter data here.
		6	units	namestr	NULL	1	0	0	Used only for columns which contain data that can be expressed as units of measure. Contains the Standard International Units of phyical quantities the data in this column should be interpreted as.
		7	label	namestr	NULL	1	0	0	Name or Brief title of the column to be used in reports. If left blank, it is filled with the column name (underscores included)
		8	description	descstr	NULL	0	0	0	Description of the function or purpose of the column
		9	null_allowed	bit	NULL	0	0	0	1 (yes) or 0 (no) indicates whether null inputs are allowed in the column being described in TOT
		10	part_of_primary_key	bit	NULL	0	0	0	1 (yes) or 0 (no) indicates if this column is part of the primary key for this table
		11	part_of_alternate_key	bit	NULL	0	0	0	1 (yes) or 0 (no) indicates if this column is part of the alternate key for this table
		12	column_order	int	NULL	0	0	0	Numerical order in which this column will be presented on a report with respect to the other columns in the table. Also the column order for multi-column keys. Should not duplicate numbers
		13	entry_date	datetimestamp	NULL	0	0	0	Date this column was entered or changed in TOT
	dataowners	1	dataowner_name	namestr	NULL	0	1	0	Name of a subsection within the database
		2	description	descstr	NULL	0	0	0	Description of the function or purpose of this section of the database

dataowner name	table name	column order	column_name	datatype	type leng	null ?	pri key	alt key	description
tot	dataowners	3	responsible_person	namestr	NULL	0	0	0	Name of the person responsible for this section of the database
		4	entry_date	datetimestamp	NULL	0	0	0	Date this dataowner was entered or changed in TOT
	datatype_defaults	1	default_name	namestr	NULL	0	1	0	The unique name of this default (must be unique compared to other datatype defaults and the column defaults)
		2	datatype	namestr	NULL	0	0	0	The datatype this default is bound to
		3	default_text	descstr	NULL	0	0	0	The Transact SQL code needed to create this default
		4	entry_date	datetimestamp	NULL	0	0	0	Date this default was entered or changed in TOT
	datatype_rules	1	rule_name	namestr	NULL	0	1	0	The unique name of this rule (must be unique compared to other datatype rules and the column rules)
		2	datatype	namestr	NULL	0	0	0	The user defined datatype this rule is bound to
		3	rule_text	descstr	NULL	0	0	0	the Transact SQL code needed to create this rule
		4	entry_date	datetimestamp	NULL	0	0	0	Date this rule was entered or changed in TOT
	datatypes	1	datatype	namestr	NULL	0	1	0	The name of the user defined datatype being described in TOT
		2	definition	namestr	NULL	0	0	0	The Transact SQL code defining this datatype
		3	description	descstr	NULL	0	0	0	Description of the function or purpose of this datatype
		4	responsible_person	namestr	NULL	0	0	0	The person who designed this datatype
		5	entry_date	datetimestamp	NULL	0	0	0	Date this datatype was entered or changed in TOT
	foreignkeys	1	foreign_dataowner_name	namestr	NULL	0	1	0	The section of the database the table the foreign key belongs to is in
		2	foreign_table_name	namestr	NULL	0	1	0	The table the foreign key belongs to
		3	foreign_column_name	namestr	NULL	0	1	0	The column that is the foreign key

dataowner name	table name	column order	column_name	datatype	type leng	null ?	pri key	alt key	description
tot	foreignkeys	4	primary_dataowner_name	namestr	NULL	0	0	0	The section of the database the primary key this foreign key is associated with resides in
		5	primary_table_name	namestr	NULL	0	0	0	The table the primary key this foriegn key is associated with resides in
		6	primary_column_name	namestr	NULL	0	0	0	The column that is the primary key this foreign key is associated with
		7	entry_date	datetimestamp	NULL	0	0	0	Date this foreign key was entered or changed in TOT
	tables	1	dataowner_name	namestr	NULL	0	1	0	The section of the database this table resides under
		2	table_name	namestr	NULL	0	1	0	Name of a table in the database being described in TOT. This should be unique within the entire database in the first 26 characters.
		3	description	descstr	NULL	0	0	0	Description of the function or purpose of the table
		4	responsible_person	namestr	NULL	0	0	0	The person who designed the table
		5	entry_date	datetimestamp	NULL	0	0	0	Date this table was entered or changed in TOT

End of Report

Foreign Table	Foreign Column	Primary Table	Primary Column	
tables	dataowner_name	dataowners	dataowner_name	
columns	dataowner_name	tables	dataowner_name	\ 2-part
columns	table_name	tables	table_name	/ primary
columns	datatype	datatypes	datatype	
column_defaults	dataowner_name	columns	dataowner_name	\
column_defaults	table_name	columns	table_name	| 3-part
column_defaults	column_name	columns	column_name	/ primary
column_rules	dataowner_name	columns	dataowner_name	\
column_rules	table_name	columns	table_name	| 3-part
column_rules	column_name	columns	column_name	/ primary
datatype_rules	datatype	datatypes	datatype	
datatype_defaults	datatype	datatypes	datatype	
foreignkeys	foreign_dataowner_name	columns	dataowner_name	\
foreignkeys	foreign_table_name	columns	table_name	| 3-part
foreignkeys	foreign_column_name	columns	column_name	/ primary
foreignkeys	primary_dataowner_name	columns	dataowner_name	\
foreignkeys	primary_table_name	columns	table_name	| 3-part
foreignkeys	primary_column_name	columns	column_name	/ primary

Definitions:

- Foreign Table - the table that has the foreign key
- Foreign Column - a column that is a part of the foreign key
- Primary Table - the table with the primary key for the foreign key being described
- Primary Column - the column of the primary key that corresponds to the associated column of the foreign key

C Appendix: TOT triggers not stored in TOT.

The TOT data schema contains the following triggers which are not specified by the definition of TOT stored within the TOT database.

The insert and update triggers are identical for all of the triggers used by TOT. This is because of the similarity of the actions in an insertion and an update.

C.1 Dataowners triggers

C.1.1 Dataowners Insert and Update Trigger

IF the dataowner being added or updated already exists in the 'dataowners' table

rollback the transaction

ELSE

set the entry date to today's date for the record to be updated

C.1.2 Dataowners Delete Trigger

IF there is an entry in the 'tables' table that is using the dataowner that is to be deleted

Give the message "This dataowner is in use"

Rollback the transaction

C.2 Tables triggers

C.2.1 Tables Insert and Update Trigger

IF the first 26 characters of the table name being inserted matches the first 26 characters of any existing table name,

Give the message "A table by this name already exists"

Rollback the transaction

ELSE IF the dataowner specified for this record does not exist

Give the message "No dataowner by this name exists"

Rollback the transaction

ELSE

set the entry date to today's date for the record to be updated

C.2.2 Tables Delete Trigger

IF there is an entry in the 'columns' table using the table to be deleted

Give the message "A table by this name has columns"

Rollback the transaction

ELSE IF there is an entry in the 'foreignkeys' table using the table to be deleted

Give the message "A table by this name has foreign keys"

Rollback the transaction

ELSE IF there is an entry in the 'group_permissions' table using the table to be deleted

Give the message "A table by this name has permissions"

Rollback the transaction

Comment: the group_permissions table is part of our application. It is coupled to TOT to help insure database integrity. This aspect of TOT is not documented elsewhere in the TOT documentation.

C.3 Columns triggers

C.3.1 Columns Insert and Update Trigger

IF the dataowner/table/column name combination being inserted already exists

Give the message "A column by this name already exists"

Rollback the transaction

ELSE IF the dataowner/table combination specified for this record does not exist in the 'tables' table

Give the message "No table by this name exists"

Rollback the transaction

ELSE

Set the entry date to today's date for the record to be updated

IF the column label field is null

Make the label the same as the column name

C.3.2 Columns Delete Trigger

IF there is an entry in the 'column_rules' table using the column to be deleted

OR there is an entry in the 'column_defaults' table using the column to be deleted

OR there is an entry in the 'foreignkeys' table using the column to be deleted

Give the message "A column by this name is in use"

Rollback the transaction

C.4 Column_Rules triggers

C.4.1 Column_Rules Insert and Update Trigger

IF the column_rule name is already in the 'datatype_rules' or 'column_rules' table

Give the message "A rule by this name already exists"

Rollback the transaction

ELSE IF the dataowner/table/column name combination specified for the rule does not exist in the 'columns' table

Give the message "There is no column by this name"

Rollback the transaction

ELSE

set the entry date to today's date for the record to be updated

C.4.2 Column_Rules Delete Trigger

No delete trigger

C.5 Column_Defaults triggers

C.5.1 Column_Defaults Insert and Update Trigger

IF the column_default name is already in the 'datatype_defaults' or 'column_defaults' table

Give the message "A default by this name already exists"

Rollback the transaction

ELSE IF the dataowner/table/column name combination specified for the default does not exist in the 'columns' table

Give the message "There is no column by this name"

Rollback the transaction

ELSE

set the entry date to today's date for the record to be updated

C.5.2 Column_Defaults Delete Trigger

No delete trigger

C.6 Foreignkeys triggers

C.6.1 Foreignkeys Insert and Update Trigger

IF the transaction has more than one row[14]

Give message "Multiple row transactions not allowed for foreignkeys"

Rollback the transaction

Do not allow any input fields to be null except entry_date

ELSE IF the dataowner/table/column name combination specified for the foreign key already exists in the 'foreignkey' table

Give message "This column already is a foreign key"

Rollback the transaction

[14] Multiple rows will not occur with the data entry portion of TOT or Data Workbench, but could be created (and must be avoided here) using isql or the SQL form in DWB.

ELSE IF the dataowner/table name combination specified for the foreign key does not exist

Give message "This table does not exist"

Rollback the transaction

ELSE IF the dataowner/table/column name combination specified for the foreign key does not exist in the 'columns' table

Give the message "There is no column by this name"

Rollback the transaction

ELSE IF the dataowner/table/column name combination specified for the primary key does not exist in the 'columns' table

Give the message "There is no column by this name"

Rollback the transaction

ELSE IF the primary key column specified is not noted as a primary key in the 'columns' table

Give the message "The primary key specified is not a primary key"

Rollback the transaction

ELSE IF the datatypes for the foreign key and primary key are not the same

OR (the lengths of the datatypes for primary and foreign keys are not equal

AND the lengths of the datatypes are not both null)

Give the message "Datatypes of foreign and primary column names do not match"

Rollback the transaction

ELSE

Set the entry date to today's date for the record to be updated

C.6.2 Foreignkeys Delete Trigger

No delete trigger

C.7 Datatypes triggers

C.7.1 Datatypes Insert and Update Trigger

IF the datatype name is already in the 'datatypes' table

Give the message "A datatype by this name already exists"

Rollback the transaction

ELSE

set the entry date to today's date for the record to be updated

C.7.2 Datatypes Delete Trigger

IF there is an entry in the 'columns' table using the datatype to be deleted

OR there is an entry in the 'datatype_rules' table using the datatype to be deleted

OR there is an entry in the 'datatype_defaults' table using the datatype to be deleted

Give the message "This datatype is in use"

Rollback the transaction

C.8 Datatype_Rules triggers

C.8.1 Datatype_Rules Insert and Update Trigger

IF the datatype_rule name is already in the 'datatype_rules' or 'column_rules' table

Give the message "A rule by this name already exists"

Rollback the transaction

ELSE IF the datatype specified for the rule does not exist in the 'datatypes' table

Give the message "There is no datatype by this name"

Rollback the transaction

ELSE

set the entry date to today's date for the record to be updated

C.8.2 Datatype_Rules Delete Trigger

No delete trigger

C.9 Datatype_Defaults triggers

C.9.1 Datatype_Defaults Insert and Update Trigger

IF the datatype_default name is already in the 'datatype_defaults' or 'column_defaults' table

Give the message "A default by this name already exists"

Rollback the transaction

ELSE IF the datatype specified for the default does not exist in the 'datatypes' table

Give the message "There is no datatype by this name"

Rollback the transaction

ELSE

set the entry date to today's date for the record to be updated

C.9.2 Datatype_Defaults Delete Trigger

No delete trigger

D Appendix: Instructions for use of the TOT data entry application

Before TOT can be run, several environmental conditions must be met. The user must have :

1. permission to use TOT

2. a login on the Sybase server
3. a login on the TOT database
4. permission to use the TOT database
    - A user who is planning on adding or updating data in TOT must have insert/update permission in the database. Select permission is required for reading.
    - A user who wishes to use the code generation portion of TOT must have write permission to the directory from which TOT is invoked.

It is assumed that the user is familiar with the commands used in the APT forms management system for moving through a form, invoking menu items, moving through fields on the form and exiting forms. See the Sybase "APT Workbench" manual for instructions.

D.1 TABLE OF TABLES FORM

The operational structure of TOT has the following hierarchy:

1. Dataowners
2. Tables
3. Columns
    - Columns
    - Defaults
    - Rules
    - Foreign Keys
    - Build Code
4. Datatypes

D.2 DATAOWNERS FORM

The dataowners form contains the following menu items and sub-menu items:

1. Find

- By Example - searches the database for data matching what has been input by the user on the form. The first matching record found is displayed on the form. If the fields that are input do not correspond to any existing data, you are told so.
- Next - finds the next instance of the data that match the fields on which you ran Find⇒By Example. If there are no more data that match this example, you are told so. If you select the Next option without previously selecting any of the Find options, all records for that table will be found.
- Previous - finds the previous instance of the data that match the fields on which you ran Find⇒By Example. If you have already backed up to the first instance that matches, you are told so.

You can also find by example using relational operators and wildcard characters. You may only do this on character fields, because the other datatypes do not accept relational operators and wildcard characters as input.

2. Do It

   - Add - allows the user to add new rows to the database. This option assumes that key fields are unique. If you try to add a row in which the primary keys already exist in the TOT database, an error message appears and the add transaction is rolled backed. See the "Modify" explanation for advice on handling this situation.
   - Modify - modifies existing rows. Modify only allows changes to non-key field values. To determine which are the key field, see Figure 1 or Appendix B . If you change a key field and then pick Modify, an error message appears. If you want to change a key field value, delete the current one and add a new row. In order to use Modify, you must first fill the form using Find⇒By Example.
   - Delete - deletes the row showing on the screen from the database. Like Add and Modify, Delete uses key field information. It checks the key fields to determine which rows to delete. If the key fields are the same as an existing row, then the row is deleted, whether or not the non-key fields are the same as the existing row.

3. Clear - just clears the form. It has no effect on the database tables.

4. Tables - calls the Tables Form and automatically fills in the dataowners_name field.

5. Special Features

   - While your cursor is positioned on the dataowner_name field, you may press CTRL-v (values key toggle). This will display a window with a list of valid dataowner_name's. The Display window may be too small to show the entire list. Use down-arrow key to see if further entries exist. Select a dataowner_name by positioning the cursor on it and pressing return will highlight it. Since this is a toggle, you must press CTRL-v again to put the selected dataowner_name in the dataowner_name field.
   - You do not have to enter a date in the entry_date field, the current date and time will automatically be put in for you when the row is Added or Modified. This will be true of any form in the TOT application where there is an entry_date field.
   - The last line on this form (and many of the other forms in TOT), is just an informational line. The number displayed after 'Rows Found:' prompt is the number of rows that matched the fields on which you ran Find⇒By Example. The 'Current Row:' number reflects the number of the row you are currently enjoying. The 'First Row in Buffer:' number is the record number of the first row still retained in the buffer.

D.3 TABLES FORM

The tables form contains the following menu items and sub-menu items:

1. Find - See the description of these options in the Dataowner section.

   - By Example
   - Next
   - Previous

2. Do It - See the description of these options in the Dataowner section.

   - Add - Since every table name must be unique within 26 characters across the entire database, no duplicate table names will be allowed to be added.

- Modify
- Delete

3. Clear - See the description of this option in the Dataowner section.

4. Columns - calls the Columns Form and automatically fills in the dataowners_name and the table_name fields.

5. Special Features

    - While your cursor is positioned on the dataowner_name or table_name fields, you may press CTRL-v (values key toggle). When you press CTRL-v while your positioned on the table_name field, only the tables that are associated with the dataowner entered in the dataowner_name field will be shown as valid tables.
    - After moving out of the dataowner_name field, if the field is not null, the data entered in that field is checked to see if it actually exists in the dataowners table.
    - After moving out of the table_name field, if the field is not null, the data entered in that field is checked to see if it actually exists in the tables table.

D.4 COLUMNS FORM

The columns form contains the following menu items and sub-menu items:

1. Find - Searches the database for data matching what has been input by the user on the form. If the fields that are filled in do not correspond to any existing data, you are told so.

2. Do It - See the description of these options in the Dataowner section.

    - Add
    - Modify
    - Delete - Same as the description for Delete in the Dataowner section, but due to the Column form showing several columns information, only the column that the cursor is positioned on will be deleted.

3. Clear - See the description of these options in the Dataowner section.

4. Scroll - Move the cursor Down or Up a row on the form.
    - Down
    - Up
5. More
    - Defaults - calls the Column Defaults form.
    - Rules - calls the Column Rules form.
    - Foreign Keys - calls the Foreign Key form
    - Build Table SQL Code
        - Build Table From Database - Builds the SQL code for the creation of a table from the description stored in the database. If the table contents entered on the form differs from that described by data stored in the columns table in the database, you will be warned and asked if you would like to modify the columns table entries with the information on the form before creating the code.
        - Build Table From Form - Builds the SQL code for the creation of a table from the information stored on the form. If you have altered the data on the form, you will be given a warning message informing you that the code created may not be consistent with what is stored in the database and then asked if you want to proceed.
6. Special Features
    - While your cursor is positioned on the dataowner_name field, table_name field or the datatype field, you may press CTRL-v (values key toggle) for a list of valid entries. When you press CTRL-v while your positioned on the table_name field, only the tables that are associated with the dataowner entered in the dataowner_name field will be shown as valid tables.
    - After moving out of the dataowner_name field, if the field is not null, the data entered in that field is checked to see if it actually exists in the dataowners table.
    - After moving out of the table_name field, if the field is not null, the data entered in that field is checked to see if it actually exists in the tables table.

- After moving out of the datatype field, if the field is not null, the data entered in that field is checked to see if it is in the systype system table. If it is a valid datatype, it is then checked to see if a length is needed. If no length is needed then the type_length field is skipped over.
- When the down arrow in the menu section is activated, you may select it to browse through the data at a faster pace than by using the Scroll⇒Down option. The same is true for the up arrow.

D.5 COLUMN DEFAULTS FORM

The columns defaults form contains the following menu items and sub-menu items:

1. Find - See the description of these options in the Dataowner section.
   - By Example
   - Next
   - Previous
2. Do It - See the description of these options in the Dataowner section.
   - Add
   - Modify
   - Delete
3. Clear - See the description of this option in the Dataowner section.
4. Special Features
   - While your cursor is positioned on the dataowner_name, table_name, or column_name fields, you may press CTRL-v (values key toggle). When you press CTRL-v while positioned on the table_name field, only the tables that are associated with the dataowner entered in the dataowner_name field will be shown as valid tables. When you press CTRL-v while positioned on the column_name field, only those columns in the table entered in the table_name field will be shown as valid columns.
   - After moving out of the dataowner_name field, if the field is not null, the data entered in that field is checked to see if it actually exists in the dataowners table.

- After moving out of the table_name field, if the field is not null, the data entered in that field is checked to see if it actually exists in the tables table.

D.6 COLUMN RULES FORM

The columns rules form contains the following menu items and sub-menu items:

1. Find - See the description of these options in the Dataowner section.
   - By Example
   - Next
   - Previous
2. Do It - See the description of these options in the Dataowner section.
   - Add
   - Modify
   - Delete
3. Clear - See the description of this option in the Dataowner section.
4. Special Features
   - While your cursor is positioned on the dataowner_name, table_name, or column_name fields, you may press CTRL-v (values key toggle). When you press CTRL-v while positioned on the table_name field, only the tables that are associated with the dataowner entered in the dataowner_name field will be shown as valid tables. When you press CTRL-v while positioned on the column_name field, only those columns in the table entered in the table_name field will be shown as valid columns.
   - After moving out of the dataowner_name field, if the field is not null, the data entered in that field is checked to see if it actually exists in the dataowners table.
   - After moving out of the table_name field, if the field is not null, the data entered in that field is checked to see if it actually exists in the tables table.

D.7 FOREIGN KEYS FORM

The foreign keys form contains the following menu items and sub-menu items:

1. Find - See the description of these options in the Dataowner section.
    - By Example
    - Next
    - Previous
2. Do It - See the description of these options in the Dataowner section.
    - Add
    - Modify
    - Delete
3. Clear - See the description of this option in the Dataowner section.
4. Build Code - See the description of these options in the Tables of Table section.
    - Fkey And Insert Trigger Code - writes the code for creating foreignkeys and for creating the insert and update trigger.
    - Build Delete Trigger Code - writes the code for creating the delete trigger.
5. Special Features
    - While your cursor is positioned on any field in this form except entry_date, you may press CTRL-v (values key toggle). When you press CTRL-v while positioned on the foreign_table_name or primary_table_name fields, only the tables that are associated with the dataowner entered in the foreign_dataowner_name or primary_dataowner_name fields will be shown as valid tables. When you press CTRL-v on either of the column_name fields, only the columns in the corresponding table will be shown as valid columns.
    - After moving out of the foreign_dataowner_name or the primary_dataowner_name fields, if the field is not null, the data entered in that field is checked to see if it actually exists in the dataowners table.

- After moving out of the foreign_table_name or the primary_table_name fields, if the field is not null, the data entered in that field is checked to see if it actually exists in the tables table.

D.8 BUILD SQL CODE FORM

The build SQL code form contains the following menu items and sub-menu items:

1. Create SQL Code
   - Build Table SQL Code - writes the SQL code for the creation of a table.
   - Fkey And Insert Trigger Code - writes the SQL code for creating foreignkeys and for creating the insert and update trigger.
   - Build Delete Trigger Code - writes the SQL code for creating the delete trigger.
   - Build All Of The Above - writes all SQL code for creating tables, foreignkeys and insert and update triggers , and delete triggers with all scripts in one file.

D.9 DATATYPES FORM

The datatypes form contains the following menu items and sub-menu items:

1. Find - See the description of these options in the Dataowner section.
   - By Example
   - Next
   - Previous
2. Do It - See the description of these options in the Dataowner section.
   - Add
   - Modify
   - Delete
3. Clear - See the description of this option in the Dataowner section.
4. More

- Create Type - this option will actually create the datatype, which can then be used in the columns table. If there were any defaults or rules defined for that datatype, it will bind them to it.
- Defaults - calls the Datatype Defaults form.
- Rules - calls the Datatype Rules form.

D.10 DATATYPE DEFAULTS FORM

The datatype defaults form contains the following menu items and sub-menu items:

1. Find - See the description of these options in the Dataowner section.
   - By Example
   - Next
   - Previous
2. Do It - See the description of these options in the Dataowner section.
   - Add
   - Modify
   - Delete
3. Clear - See the description of this option in the Dataowner section.
4. Special Features
   - While your cursor is positioned on the datatype field, you may press CTRL-v (values key toggle). This will display a window with a list of valid datatypes.

D.11 DATATYPE RULES FORM

The datatype rules form contains the following menu items and sub-menu items:

1. Find - See the description of these options in the Dataowner section.
   - By Example
   - Next

- Previous

2. Do It - See the description of these options in the Dataowner section.
   - Add
   - Modify
   - Delete
3. Clear - See the description of this option in the Dataowner section.
4. Special Features
   - While your cursor is positioned on the datatype field, you may press CTRL-v (values key toggle). This will display a window with a list of valid datatypes.

E Appendix: Instructions for TOT Build Applications

E.1 BUILDTABLE

The Buildtable procedure produces a SQL script for creating the table, the primary key, the primary index, the alternate index (if needed) and the table permissions.

* This procedure can be invoked when you select one of the following:
  - 'Build Table Sql Code' item under the 'Create Sql Code' option in the 'BUILD SQL CODE' form.
  - 'Build Table Sql Code' item under the 'More' option in the 'COLUMNS' form. Under this item you are given the option of either building the code based on what is stored in the database or based on what is currently stored on the form. If you build the code based on what is currently stored on the form, you will be given a message warning you that the code created may no be consisted with what is stored in the database.
* Output: " 'dataowner_name'_'table_name'.sql" - stored in the directory that TOT was invoked from.
* Actual Code:

1. The database name 'mtfmsa' is hardcoded. In a future release, the current database name should be used.

2. Before a table is created, first drop the existing table. The syntax for dropping a table is as follows:

   DROP TABLE [[database.]owner.]table_name

3. Create table:

   CREATE TABLE [[database.]owner.]table_name
   (column_name datatype [NOT NULL | NULL]
   [, column_name datatype [NOT NULL | NULL]]...)

   To do this we create a dynamic string for each column row, consisting of 'column_name datatype[(type_length)] [NULL]'. The following rules are checked against each column's information:

   IF the field 'type_length' is not null THEN append the type_length to the dynamic string.

   IF the bit field 'null_allowed' is set THEN append NULL to the dynamic string.

   IF the bit field 'part_of_primary_key' is set THEN append the column name to the dynamic string that is being built for the primary key. The syntax of this dynamic string will be shown later.

4. Create primary key:

   EXECUTE sp_primarykey table_name, col1 [, col2, col3, ..., col8]

   'col1' is the first column that makes up the primary key. The primary key can consist of one to eight columns. Note: The column order within the key is specified by the 'column_order' field in the 'columns' table.

5. Grant permissions:

   GRANT [SELECT | INSERT | DELETE | UPDATE] ON table_name TO name_list

   REVOKE [SELECT | INSERT | DELETE | UPDATE] ON table_name FROM name_list

   'name_list' can be either Sybase users name or group_name. At present, SELECT is GRANTed to the public, INSERT is GRANTed to the groups (administrator, developer, test_manager, measurer,

data_clerk), DELETE is GRANTed to the groups (administrator and developer), and UPDATE is GRANTed to the groups (administrator and developer).

6. Create a clustered index on the primary key:
   - CREATE CLUSTERED INDEX sn_index ON 'dataowner_name'.'table_name' (pkey)

   'pkey' is the primary key for the table entered on the form. This key is found querying the columns table to find the column(s) that have the bit field 'part_of_primary_key' set for this table. See above note regarding column order.

7. Create a unique nonclustered index on the alternate key:
   - CREATE UNIQUE NONCLUSTERED INDEX alt_key_index ON 'dataowner_name'.'table_name' (alt_key [, alt_key]...)

   'alt_key' is a single or multi-column key that uniquely identifies a row in this table. By making this a UNIQUE index, we instruct Sybase to check for duplicate rows, thus eliminating the need for creating trigger code to do this. Column order in the key is specified by the column_order field of the columns table as for the primary key.

8. Add column rules, if needed. See Appendix E.4
9. Add column defaults, if needed. See Appendix E.5

E.2 BUILDFKEY

The Buildfkey procedure produces two SQL scripts. One script creates the foreign keys and creates indices on those foreign keys and the other script creates the insert and update trigger as one trigger. The trigger is created to insure the integrity of the foreign keys. Also, for every distinct table that a foreign key points to, a new delete trigger code will be generated and appended to the 'dataowner_name'_'table_name'_fkey.sql' code that this procedure creates.

* This procedure is invoked when you select one of the following:
  - 'Fkey And Insert Trigger Code' item under the 'Create Sql Code' option in the 'BUILD SQL CODE' form.
  - 'Fkey And Insert Trigger Code' item under the 'More' option in the 'FOREIGNKEYS' form.

* Output:
  " 'dataowner_name'_'table_name'_fkey.sql" and
  " 'dataowner_name'_'table_name'_trig.sql"
  - stored in the directory that TOT was invoked from. The code created is based on the data that is stored in the database. This should be run after you are done entering all of the information for the table that has the foreign keys and the tables that the foreign keys point to.

* Actual Code: - " 'dataowner_name'_'table_name'_fkey.sql"

  1. Since all of our application tables are stored in the 'mtfmsa' database, our first step is to hardcode which database to use.
  2. Before foreign keys are created, existing foreign keys between the two tables are dropped. The syntax for dropping a foreign_key is as follows:
     - EXECUTE sp_dropkey foreign, foreign_table_name, deptabname

     'foreign_table_name' is the name of the table that contains the foreign key to be dropped.
     'deptabname' is the name of the dependent table (in the form of " 'dataowner_name'.'table_name' ") that contains the column(s) that the foreign key points to.
     For each dependent table that this table's foreign keys point to, an EXECUTE sp_dropkey line is created.
  3. EXECUTE 'sp_foreignkey' to add foreign key for each new one required.
  4. Create index:
     - CREATE INDEX fk#_index ON 'dataowner_name'.'table_name' (fkey) 'fkey' is the foreign key for the table entered on the form.

* Actual Code: - " 'dataowner_name'_'table_name'_trig.Sql"

  1. Since all of our application tables are stored in the 'mtfmsa' database, our first step is to hardcode which database to use.
  2. Before a trigger is created, first drop any existing trigger. The syntax for dropping a trigger is as follows:
     - DROP TRIGGER trigger_name

'trigger_name' is the name of the trigger.

3. Create trigger:

   CREATE TRIGGER trigger_name ON table_name FOR INSERT, UPDATE AS
   BEGIN
   .
   .
   .
   END

   'trigger_name' is the name of the trigger. The trigger name has the following format, " 'table_name'_i". Since trigger names can only be 30 characters or less, if necessary the 'table_name' is trimmed to 26 characters.

* Body of trigger:

1. Declare variables.
2. Check to see if there were any rows entered.
3. If the primary key is of sntype and is not a foreign key, make sure that it is not updated or inserted.
4. For each foreign key row, check to see if the primary key that the foreign key is attached to exists. IF the primary key does not exist THEN set the 'problem' flag.

   Since some foreign keys are allowed to be NULL, the following rules are checked:

   - IF there are any rows being inserted or updated with NULLs for this field THEN ignore them.
   - To allow for multi row inserts, only check the ones that are not NULL. IF a primary key is not found for any of these THEN rollback the whole transaction.
5. Check to see IF the 'problem' flag was set THEN print error message and exit the trigger.
6. For each record inserted for which the primary key is a single column of type sntype and for which the primary key is not a foreign key to another table, generate a serial number[15].

[15]This is done with a stored procedure which assigns a generated serial number to the record that has a NULL primary_key.

E.3 BLDDELTRIG

The Blddeltrig procedure produces a SQL script which creates the delete trigger. The trigger is created to ensure that no primary key value in this table can be deleted if a foreign keys exist that points to it.

* This procedure is invoked when you select one of the following:
  - 'Build Delete Trigger Code' item under the 'Create Sql Code' option in the 'BUILD SQL CODE' form.
  - 'Build Delete Trigger Code' item under the 'More' option in the 'FOREIGNKEYS' form.

* Output:
  " 'dataowner_name'_'table_name'_del_trig.sql". - stored in the directory that TOT was invoked from. The code created is based on the data that is stored in the database. This should be run, for the primary key table, every time that a new foreign key is created which points to the primary key in this table.

* Actual Code:

  1. Since all of our application tables are stored in the 'mtfmsa' database, our first step is to hardcode which database to use.
  2. Before a trigger is created, the convention to use is to drop the trigger first. The syntax for dropping a trigger is as follows:
     - DROP TRIGGER trigger_name
       'trigger_name' is the name of the trigger.
     - Create trigger:
       * CREATE TRIGGER trigger_name ON table_name FOR DELETE AS
       * BEGIN
       * .
       * .
       * .
       * END

       'trigger_name' is the name of the trigger. The trigger name has the following format, " 'table_name'_d". Since trigger names can only be 30 characters or less, if necessary the 'table_name' is trimmed to 26 characters.

* Body of trigger:

  1. Declare variables.
  2. Check to see if any rows are to be deleted.
  3. For each row being deleted, check each table which has a foreign key pointing at (the primary key of) this table. IF there are entries in any of them THEN print error message and roll back this transaction.

E.4 ADD Column RULE

* This procedure is invoked as part of creating the code for building a table.

* Output: " 'dataowner_name'_'table_name'_crule.sql" - stored in the directory that TOT was invoked from.

* Actual Code:

  1. Before a rule is added to a column, one must drop the rule for that column first. Before dropping a rule one must un-bind all references to it. The syntax for dropping a rule is:
     - DROP RULE rule_name
  2. Create rule:
     - CREATE RULE " 'dataowner_name'.'rule_name' "
  3. Bind the rule to the column:
     - EXECUTE sp_bindrule rule_name,
       " 'table_name'.'column_name' " AS rule_text

E.5 ADD Column DEFAULT

* This procedure is invoked as part of creating the code for building a table.

* Output: " 'dataowner_name'_'table_name'_cdefault.sql" - stored in the directory that TOT was invoked from.

* Actual Code:

1. Before a default is added to a column, one must drop the default for that column first. Before dropping a default one must un-bind all references to it. The syntax for dropping a default is:
   DROP DEFAULT default_name
2. Create default:
   CREATE DEFAULT " 'dataowner_name'.'default_name' "
3. Bind the default to the column:
   - EXECUTE sp_binddefault default_name,
     " 'table_name'.'column_name' " AS default_text

E.6 ADDTYPE

The Addtype procedure is different in that is does not produce a SQL script. The adding of a datatype is processed as soon as you execute the 'Create type' item.

* This procedure is invoked when you select the 'Create Type' item under the 'More' option in the 'DATATYPES' form. This directly submits an EXECUTE sp_addtype to the server. For best results:

  Add the current form information into the database before invoking 'Create Type'. Do this by selecting option 2, item 'add'.

  Clear the form. Select option 3.

  Find the information from the database of the type you want to add.

  Enter the datatype on the form and select 'By Example' item of option 1.

  Create type. Select the 'Create Type' item under option 4.

* Output: none

* Actual Code:

1. Before a type is added, any existing definition must be dropped. The syntax for dropping a datatype is as follows:
   - EXECUTE sp_droptype datatype
2. Create datatype:
   - EXECUTE sp_addtype datatype, definition

'definition' is the physical or SQL Server-supplied type (char, int, etc.) on which the user-defined datatype is based.

3. Add datatype rule, if needed. See Appendix E.7
4. Add datatype default, if needed. See Appendix E.8

E.7 ADD Datatype RULE

* This procedure is invoked as part of adding a user-defined datatype. This directly submits DROP RULE, CREATE RULE and EXECUTE sp_bindrule for each datatype stored in the 'datatype_rules' table.

* Output: none

* Actual Code:

1. Before a rule is added to a datatype, one must drop the rule for that datatype first. Before dropping a rule one must un-bind all references to it. The syntax for dropping a default is:
   - DROP RULE rule_name
2. Create rule:
   - CREATE RULE rule_name
3. Bind the rule to the datatype:
   - EXECUTE sp_bindrule rule_name, datatype, futureonly AS rule_text

   'futureonly' prevents existing columns of a user-defined datatype from inheriting the new rule.

E.8 ADD Datatype DEFAULT

* This procedure is invoked as part of adding a user-defined datatype. This directly submits DROP DEFAULT, CREATE DEFAULT and EXECUTE sp_binddefault for each datatype
stored in the 'datatype_defaults' table.

* Output: none

* Actual Code:

1. Before a default is added to a datatype, one must drop the default for that datatype first. Before dropping a default one must unbind all references to it. The syntax for dropping a default is:
   - DROP DEFAULT default_name
2. Create default:
   - CREATE DEFAULT default_name
3. Bind the default to the datatype:
   - EXECUTE sp_binddefault default_name, datatype, futureonly AS default_text

   'futureonly' prevents existing columns of a user-defined datatype from inheriting the new rule.

F Appendix: Primary, Alternate and Foreign key options available using TOT

The use of keys in a database schema allow one to specify one or more columns within a table which must be unique among all rows entered in that table. A primary key is the column or set of columns which serves as the principle unique specifier for that table. In the case of tables where one chooses to create a serial number which serves as the unique key, there must be some data stored in other columns which specify uniquely the values which are of interest in that row. This is identified as the alternate key. When one wishes to maintain data integrity by demanding that a value in a table be one which has previously been entered into another table, a Foreign Key describes that relation. SYBASE maintains a foreignkeys table, but that guarantees no relational integrity. To maintain integrity, one needs SYBASE triggers.

TOT allows one to automatically generate the code for keys. Some features of this are:

1. TOT requires a primary key. It can consist of one or more columns. This is enforced in BUILDTABLE.

2. For a primary key with multiple columns, TOT will create the key using 'sp_primarykey' with the order specified in the 'column_order' column. To utilize this as a foreign key, one will need to create trigger code outside of TOT.

3. For a foreign key consisting of a single column, in addition to having the foreign key declared automatically, INSERT, UPDATE and DELETE triggers will be created to enforce referential integrity.

4. If the primary key is a serial number (datatype = sntype) then the code will be written to cause that serial number to be automatically generated by the insert trigger UNLESS that key is a foreign key to another table. BUILDFKEY will report an error if it writes serial number generation code but there is no alternate key specified for that table.

5. One may have tables in which no alternate key is specified. No Relational Model rule requires more than one key in a table.

In Section 2 the indices which are created for each key are described. Additional detailed information on triggers created by TOT is contained in Appendix E

G Appendix: Defaults and Rules

Defaults and rules are two tools that Sybase offers the system designer for guiding the use of a table. A default value is one that is put into a column when a row is inserted without specifying a value for that column. A rule is a restriction of the values allowed to be inserted in a column.

A default is an expression composed of constants and functions of constants. See "Expressions" in the Sybase Commands Manual. Depending on the data type of the column it might be an integer (-1), a datetime (getdate()). or a character string (user_name()+'did not put anything here'). To use a default you must first create it, giving it a name. See "CREATE DEFAULT" in the Sybase Commands Reference. The default can then be bound to a specific column in a table. A default can also be bound to a user-defined datatype. Any further tables created thereafter using that datatype will have that default bound to each column of that datatype. A default may be bound to more than one datatype or default or both. See "sp_bindefault" in the Sybase Commands Reference.

A rule is a logical expression involving constants, functions, operators, and at most one local variable. See "Expressions" in the Sybase Commands Manual. If that expression does not evaluate to TRUE when a row is inserted, then the row is rejected. The value that you are attempting to insert is substituted for the local variable when the expression is evaluated. For

example, you can require (@my_integer > 3) or (@made_date < get_date()). Rules must be created. See "CREATE RULE" in the SYBASE Commands Manual. They can then be bound to either columns or user datatypes or both, just as defaults can. See "sp_bindrule" in the Sybase Commands Manual.

To provide a more complicated default or rule you must use a trigger. Right now we have no rules defined for the system. The only default we have defined is for the datetime stamp. Our datetime datatype differs from the system datetime datetimestamp only in having the default of the current time.

In managing these entities one must be quite careful. If you change a rule or default you must bind the new version to everything that has the old binding.

H Appendix: Instructions for installation of TOT.

A complete package to install TOT requires a set of instructions plus the following items:

1. Transact-SQL script to create a suitable set of default data types
2. Transact-SQL script to create the TOT tables
3. Transact-SQL scripts to create the TOT triggers
4. A script plus data files to enter the TOT schema into the TOT tables via bulk copy
5. Sybase APT-SQL code (forms and application code) for the TOT data entry and TOT BUILD procedures.

If one also wishes to utilize the serial number system for which TOT can generate automatic triggers, the above list should be augmented with

1. A script plus data files to enter the Serials schema into the TOT tables via bulk copy
   OR
2. Transact-SQL code created by TOT to create these tables.
3. A Transact-SQL script for creating the Serial Generator stored procedure.

An installation package for TOT is not available at this time.

TOT data entry and BUILD applications run under SYBASE APT-Execute. To modify the APT-SQL code one needs APT-Workbench.

References

[1] J. Harvey Trimble Jr and David Chappell. *A Visual Introduction to SQL.* John Wiley and Sons, New York, 1989.

[2] Mark A. Linton, Paul R. Calder, and John M. Vlissides. Interviews: A c++ graphical interface toolkit. Technical Report CSL-TR-88-358, Stanford University, July 1988.

[3] John M. Vlissides and Mark A. Linton. Applying object-oriented design to structured graphics. In *Proceeding of the 1988 USENIX C++ Conference*, pages 81–94, October 1988.